FACULTY OF INFORMATICS
COURSEWORK COVERSHEET

SUBJECT'S INFORMATION:			
Subject:	CSCI361 Cryptography and Secure Applications		
Session:	February 2021		
Programme / Section:	BCS (S1, D1)		
Lecturer:	**Mohamad Faizal Alias**		
Coursework Type *(tick appropriate box)*	❑ Individual Assignment		
Coursework Title:	Assignment 4	Coursework Percentage:	10%
Hand-out Date:	Week 12	Received By : (signature)	
Due Date:	Week 16	Received Date :	
STUDENT'S INFORMATION:			
Student's Name & ID:	**Lee De Bin** **6702934**		
Contact Number / Email:	**j16022487@student.newinti.edu.my**		
STUDENT'S DECLARATION			
By signing this, I / We declare that: 1. This assignment meets all the requirements for the subject as detailed in the relevant Subject Outline, which I/ we have read. 2. It is my / our own work and I / we did not collaborate with or copy from others. 3. I / we have read and understand my responsibilities under the University of Wollongong's policy on plagiarism. 4. I / we have not plagiarised from published work (including the internet). Where I have used the work from others, I / we have referenced it in the text and provided a reference list at the end of the assignment. I am / we are aware that late submission without an authorised extension from the subject co-ordinator may incur a penalty. *(See your subject outline for further information).*			
Name & Signature:	Lee De Bin		

COURSEWORK SUBMISSION RECEIPT

Subject:	CSCI361 Cryptography and Secure Application	Session:	February 2021
Programme / Section:	BCS	Lecturer:	**Mohamad Faizal Alias**
Coursework Type: (Tick appropriate box)	❑ Individual Assignment		
Coursework Title:	Assignment 4	Coursework Percentage:	10%
Hand-out Date:	Week 12	Received By: (Signature)	
Due date:	Week 16	Received Date:	
STUDENT'S INFORMATION:			
Student's Name & ID:	Lee De Bin 6702934		
Contact Number / Email:	0129081390		

Assessment Criteria		**Total Marks**	**Given Marks**
1.	Client-Server Execution	10	
2.	Client - Key Generation coding **RSA (PUa & PRa)**	10	
3.	Server Verify **PUa** with Hash of **PUa** – received from Client	10	
4.	Server IDEA Key Gen (**Ks**) & SHA-1 Hash for **Ks + E(PUa, Ks)**	20	
5.	Client decrypt **D(PRa, Ks)** and Verify Hash of Ks	10	
6.	Encrypted data communication process – **IDEA with CFB Mode**	20	
7.	Overall report & Presentation	20	
		100	
		Penalty	
Marked by: _____ Date: _____		**Final Mark (10 %)**	

Lecturer's Comments

Penalty for late submission:

1 day – minus 20% of total mark awarded
2 days – minus 50% of total mark awarded
3 days – 0 mark for this piece of coursework

University of Wollongong
CSCI361: Cryptography and Secure Applications
February 2021 Session
Individual Assignment 4 (10 %)

Tasks:

- Extend and implement of the **RSA** PKC scheme in station-to-station communication
- Using Hashing for integrity of message, that is **SHA-1**
- Produce simple Key Transport protocol
- Encrypt Key with **IDEA** encryption
- Mode of Block Cipher is **CFB Mode**

Specifications:

The aim of this assignment is to create a program that allows two different stations to communicate by initiating the following protocols:

Some Guidelines before you start:

1. Those who planned to use VM or VirtualBox, make sure that you've installed VM/VirtualBox and configure the Network and set the configuration to use Bridge mode
2. After setting up VM/VirtualBox with Bridge mode only install your Linux or other OSes that you preferred.
3. Each time you test the server and the client program, the server need to open a port (port number). Any error during testing your OS might not release the port number and/or the port currently having buffer of previous data. This might lead to error on your second-time execution of your program. It is suggested that each time you execute server, change the port number and let the client program use the new port number opened.
4. Developed both server and client code in separate folders. This to ensure any local copy of reference file (if so exists) won't give you a misleading error or/and success of the code execution.
5. Use any C++ sample of socket programming and execute them first to ensure that the sample is working on real network environment. It is NOT suggested to use WinSock since it might bound to restriction in Windows environment.
6. You can use two computers and connect them via Ethernet cable (CAT5 cable) directly to simulate a network.
7. You may be interested to venture into multithreading
8. Observed carefully the protocol given below:

Handshake Process:

 Client Program	Preferable your program is proven to be running on the real network NOT only on Localhost with VMWare 	 Server Program
		1. Start a Host (server) – Initiate by entering the port no. for client to connect
2. Execute client – enters the IP and Port number of the Server 3. Client program: starts with generating key pairs; PU_A-Public Key and PR_A-Private Key using **RSA** 4. PU_A will be send over to the server, together with the Hash of PU_A using **SHA-1** for integrity checking. PR_A is kept by Client for later Decryption.	PUa || H(PUa) || Flag to connect (optional) 	5. Upon receiving PU_A and $H(PU_A)$, Verify them using **SHA-1**. 6. Server generates a session key K_s (**IDEA Key size)** for the purpose of **IDEA encryption**
	E(PUa, Ks) || H(Ks) || Flag for acknowledgement (optional) 	7. Server then sends over to Client, encrypted K_S using PU_A that is $E(PU_A, K_S)$ and $H(K_S)$
8. Upon Receiving of $E(PU_A, K_S)$ and $H(K_S)$; Client Decrypt the message using PR_A , get to know K_s (**IDEA Key size)**. 9. Then it verifies the integrity of K_s by checking $H(K_s)$ – Hashing via **SHA-1**		

Note:

- Prepare your program so that it will request the user to enter appropriate settings such as IP number and port number.
- Connection is using socket programming either using UDP or TCP.
- You may use all functionalities available in Crypto++ library
- Flags are optional depending on your technique to control the handshake process.

Data Communication Process:

Client Program		Server Program
10. Upon completion of Handshake process above, Client is now ready to use K_S to encrypt its data communication 11. Client program: Encrypt the message **Mc** using **IDEA** encryption with the agreed session **Ks** (**IDEA Key size**) 12. Message **Mc** enters **IDEA** encryption using **CFB Mode**	E(Ks, M) || Flag – from Client 	13. Server then Decrypt the message **Mc** from client using **Ks** 14. Server program: Encrypt the message **Ms** using **IDEA** encryption with the agreed session **Ks** (**IDEA Key size**) **15.** Message **Ms** enters **IDEA** encryption using **CFB Mode**
17. Upon receiving message **Ms** from server, client can decrypt it using the session **Ks**. 18. Communication continues until '**Quit'** signal initiated	E(Ks, M)|| Flag – from Server 	16. Communication continues until '**Quit'** signal initiated

Note:

- The working of message sending and receiving works something like a chat program.
- All the above decisions and reusable library must be reported and reference accordingly
- Data communication process between A and B can be done either through VMware or a more appropriate approach is over the real network (you can use 2 computers for this purpose during presentation)
- Flags are optional – depending on how you control the data communication

Reporting

You are to include in your report all the following (but not bound to only these requirements):

1. Setting-up of the Stations involved simulation requirements etc.
2. All cryptosystem and Hashing strategies implemented
3. Discussion on the execution (steps) of your program
4. program explanation (on all methods used), overall program structure, data input/output, analysis results
5. Other requirements deem important

Submission

Your source code should be submitted together with the report.pdf file in one folder. ZIP the folder and named it <your-name-CSC361-Assign4>.zip and submit this ZIP file to Moodle Submission link provided. Remember to put your name and student number in all source codes (comments header).

Provide readme.txt file to guide me on your library used, execution, setting up of IP number and port examples; and/or other deemed important. Make sure the folder also has error free compiled version of your program(s).

Assignments must be submitted electronically via Moodle submission link.

Presentation

A short presentation of your working program is required. Presentation slot will be announced by your lecturer during your face-to-face class session and will be conducted during lab session of Week 16 and/or Week 17.

Extra Notes:

Depending on COVID-19 situation you may need to prepare your presentation video and upload to YouTube and share the link in you report and/or inside the readme file.

Video presentation should demonstrate the working of 2 PCs communicating securely on the network. After all operations shown and explained, continue explain your code segments.

Plagiarism

A plagiarised assignment will receive a zero mark (and penalised according to the university rules). Plagiarism detection software will be used.

Simulation requirements

(Server Side)

OS: Linux /

Oracle VM VirtualBox (Linux OS)

Network setting: Bridged Adapter

Language: C++

Library used: Crypto++® Library 8.5

Link: https://www.cryptopp.com/

(Client Side)

OS: Linux /

Oracle VM VirtualBox (Linux OS)

Network setting: Bridged Adapter

Language: C++

Library used: Crypto++® Library 8.5

Link: https://www.cryptopp.com/

1. Start a Host (server) – Initiate by entering the port no. for client to connect (**Server side**)

```
int main(int argc, char const* argv[])
{
    string RecieveMsg = "Server Received.";
    string returnSessionKey;

    int new_socket = socket(), valread;
```

```
int socket()
{
    int PORT;
    do {
        cout << "Enter port number to start the listener : ";
        cin >> PORT;
        if (PORT > 65535 || PORT < 1)
        {
            cout << "Please try again. The port range is 1 - 65535." << endl;
        }
    } while (PORT > 65535 || PORT < 1);
    printf("[Server] Successfully listening to port %d\n", PORT);
    cout << "[Server] Waiting for client to connect..." << endl;
    int server_fd, new_socket, valread;
    struct sockaddr_in address;
    int opt = 1;
    int addrlen = sizeof(address);
    // Creating socket file descriptor
    if ((server_fd = socket(AF_INET, SOCK_STREAM, 0)) == 0)
    {
        perror("socket failed");
        exit(EXIT_FAILURE);
    }

    // Forcefully attaching socket
    if (setsockopt(server_fd, SOL_SOCKET, SO_REUSEADDR | SO_REUSEPORT,
        &opt, sizeof(opt)))
    {
        perror("setsockopt");
        exit(EXIT_FAILURE);
    }
    address.sin_family = AF_INET;
    address.sin_addr.s_addr = INADDR_ANY;
    address.sin_port = htons(PORT);

    // Forcefully attaching socket
    if (bind(server_fd, (struct sockaddr*)&address,
        sizeof(address)) < 0)
    {
        perror("bind failed");
        exit(EXIT_FAILURE);
    }
    if (listen(server_fd, 3) < 0)
    {
        perror("listen");
        exit(EXIT_FAILURE);
    }
    if ((new_socket = accept(server_fd, (struct sockaddr*)&address,
        (socklen_t*)&addrlen)) < 0)
    {
        perror("accept");
        exit(EXIT_FAILURE);
    }
    return new_socket;
}
```

2. Execute client – enters the IP and Port number of the Server (**Client side**)

```
int socket()
{
    cout << "Enter the Server IP Address : ";
    string ip;
    cin >> ip;
    int PORT;
    do {
        cout << "Enter the Port Number : ";
        cin >> PORT;
        if (PORT > 65535 || PORT < 1)
        {
            cout << "Please try again. The port range is 1 - 65535." << endl;
        }
    } while (PORT > 65535 || PORT < 1);
    int sock = 0, valread;
    struct sockaddr_in serv_addr;
    if ((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0)
    {
        printf("\n Socket creation error \n");
        exit(0);
        return -1;
    }

    serv_addr.sin_family = AF_INET;
    serv_addr.sin_port = htons(PORT);

    // Convert IPv4 and IPv6 addresses from text to binary form
    if (inet_pton(AF_INET, ip.c_str(), &serv_addr.sin_addr) <= 0)
    {
        printf("\nInvalid address/Address not supported \n");
        exit(0);
        return -1;
    }

    if (connect(sock, (struct sockaddr*)&serv_addr, sizeof(serv_addr)) < 0)
    {
        printf("\nConnection Failed \n");
        exit(0);
        return -1;
    }
    return sock;
}
```

3. Client program: starts with generating key pairs; PU_A-Public Key and PR_A-Private Key using **RSA (Client side)**

```
void keyGen()
{
    AutoSeededRandomPool rng;
    InvertibleRSAFunction privkey;
    privkey.Initialize(rng, 1024);

    // Generate Private Key
    RSA::PrivateKey privateKey;
    privateKey.GenerateRandomWithKeySize(rng, 1024);
    // Generate Public Key
    RSA::PublicKey publicKey;
    publicKey.AssignFrom(privateKey);
    SaveHexPublicKey("PublicKey.txt", publicKey);
    SaveHexPrivateKey("PrivateKey.txt", privateKey);
}

void Save(const string& filename, const BufferedTransformation& bt)
{
    FileSink file(filename.c_str());
    bt.CopyTo(file);
    file.MessageEnd();
}

void SaveHex(const string& filename, const BufferedTransformation& bt)
{
    HexEncoder encoder;
    bt.CopyTo(encoder);
    encoder.MessageEnd();
    Save(filename, encoder);
}

void SaveHexPrivateKey(const string& filename, const PrivateKey& key)
{
    ByteQueue queue;
    key.Save(queue);
    SaveHex(filename, queue);
}

void SaveHexPublicKey(const string& filename, const PublicKey& key)
{
    ByteQueue queue;
    key.Save(queue);
    SaveHex(filename, queue);
}
```

4. PU_A will be send over to the server, together with the Hash of PU_A using **SHA-1** for integrity checking. PR_A is kept by Client for later Decryption. (**Client side**)

```
send_recv(sock, publickey, "Client --> Server | Public Key is Sent to Server");

string SHA1PublicKey = SHA1string(publickey);
cout << "[Client] SHA-1 Hash of Public Key : " << SHA1PublicKey << endl;

send_recv(sock, SHA1PublicKey, "Client --> Server | SHA-1 Hash Value of Public Key is Sent to Server");

cout << "Receiving Encrypted Session Key and Its Hashing Value From Server..." << endl << endl;
```

Function to communicate with server.

```
string send_recv(int socket, string message, string comments)
{
    int valread;
    char buffer[1024] = { 0 };
    send(socket, message.c_str(), strlen(message.c_str()) + 1, 0);
    cout << "[Client] Waiting/Receiving Message from Server... " << endl;
    valread = read(socket, buffer, 1024);
    cout << "[Message from Server] : ";
    printf("%s\n", buffer);
    cout << "[Details] " << comments << endl << endl;
    hashEncodedSession = buffer;
    encryptedmsg = buffer;
    return buffer;
}
```

SHA-1 hashing function

```
string SHA1string(string sha1)
{
    byte digest[SHA1::DIGESTSIZE];
    string message = sha1;
    SHA1 hash;
    hash.CalculateDigest(digest, (const byte*)message.c_str(), message.length());
    HexEncoder encoder;
    string output;
    encoder.Attach(new CryptoPP::StringSink(output));
    encoder.Put(digest, sizeof(digest));
    encoder.MessageEnd();
    return output;
}
```

5. Upon receiving PU_A and $H(PU_A)$, Verify them using **SHA-1**. (**Server side**)
   Store the public key and its hash value from client to variable. After that called the “verify” function to verify the client public key and hash value using server own sha-1 function.

```
string publickey = send_recv(new_socket, RecieveMsg, "Public Key from Client");

string hashvalue = send_recv(new_socket, RecieveMsg, "Public Key Hash Value from Client");

verify(hashvalue, SHA1string(publickey));
```

Function to communicate with client.

```
string send_recv(int new_socket, string message, string comments)
{
    int valread;
    char buffer[1024] = { 0 };
    valread = read(new_socket, buffer, 1024);
    cout << "[Server] Receiving/Waiting Message from Client..." << endl;
    cout << "[Message from Client] : " << buffer << endl;
    send(new_socket, message.c_str(), strlen(message.c_str()) + 1, 0);
    cout << "[Details] " << comments << endl << endl;
    return buffer;
}
```

SHA-1 Hashing function (client public key to verify)

```
string SHA1string(string sha1)
{
    byte digest[SHA1::DIGESTSIZE];
    string message = sha1;
    SHA1 hash;
    hash.CalculateDigest(digest, (const byte*)message.c_str(), message.length());
    HexEncoder encoder;
    string output;
    encoder.Attach(new StringSink(output));
    encoder.Put(digest, sizeof(digest));
    encoder.MessageEnd();
    return output;
}
```

Verify function (compare both string) (the public key hash value from client and the server hashing of client public key value.)

```
void verify(string a, string b)
{
    int result = strcmp(a.c_str(), b.c_str());
    cout << "[Server] Verifying the Public Key..." << endl << endl;
    if (result == 0)
    {
        cout << "[Server] Public Key is Matched!" << endl << endl;
    }
    else
    {
        cout << "[Server] Public key is Not Match!" << endl << endl;
        exit(0);
    }
}
```

6. Server generates a session key **K_s (IDEA Key size)** for the purpose of **IDEA encryption** (**Server side**)

   Server generates session key function. This function will generate the session key with random block and IDEA key size. Next it will encrypt the session key with the client public key.

```
string generateSessionKey(int sock, string publickey)
{
    AutoSeededRandomPool prng;
    InvertibleRSAFunction parameters;
    RSA::PublicKey publicKey(parameters);
    parameters.GenerateRandomWithKeySize(prng, 1024);
    SecByteBlock key(IDEA::DEFAULT_KEYLENGTH);
    prng.GenerateBlock(key, key.size());

    //Convert key from bytes to string
    string stringKey, temporary;
    ArraySource(key, sizeof(key), true, new StringSink(stringKey));
    string encodestringKey;
    StringSource encodekey(stringKey, true, new HexEncoder(
        new StringSink(temporary)));
    encodestringKey = temporary.substr(0, 32);
    cout << "[Server] Generating Session Key...\n\n";
    cout << "[Server] Session Key : " << encodestringKey << endl << endl;

    IDEAkey = encodestringKey;
    SessionKeyHashValue = encodestringKey;

    string decodedpubkey;
    StringSource decodekey(publickey, true, new HexDecoder(new StringSink(decodedpubkey)));
    StringSource pubKeySS(decodedpubkey, true);
    publicKey.Load(pubKeySS);

    string encryptedSessionKey;
    RSAES_OAEP_SHA_Encryptor e(publicKey);
    StringSource encryptboth(encodestringKey, true, new PK_EncryptorFilter(prng, e, (new HexEncoder(new StringSink(encryptedSessionKey)))));

    cout << "[Server] Encrypting Session Key with Public Key...\n\n";
    cout << "[Server] Encrypted Session Key : " << encryptedSessionKey << endl << endl;

    return encryptedSessionKey;
}
```

7. Server then sends over to Client, encrypted K_S using PU_A that is $E(PU_A, K_S)$ and $H(K_S)$ (**Server side**)

   Server send the encrypted session key and its hash value. Firstly store the generated session key to “returnSessionKey” variable by using the generateSessionKey from above. Next, send to client via send_recv function. Convert the session key into sha-1 hash using the ‘SHA1String’ function from above and send it to client via ‘send_recv’.

```
    returnSessionKey = generateSessionKey(new_socket, publickey);

    send_recv(new_socket, returnSessionKey, "Server --> Client | Encrypted Session Key is Sent to Client");

    string SHA1SessionEncode = SHA1string(SessionKeyHashValue);

    cout << "[Server] Generating Hash Value Of Encrypted Session Key...\n\n";
    cout << "[Server] SHA-1 Hash of Session Key : " << SHA1SessionEncode << endl << endl;
    returnSessionKey = send_recv(new_socket, SHA1SessionEncode, "Server --> Client | SHA-1 Hash Value of Session Key is Sent to Client");
```

8. Upon Receiving of $E(PU_A, K_S)$ and $H(K_S)$; Client Decrypt the message using PR_A , get to know **K_s (IDEA Key size). (Client side)**
9. Then it verifies the integrity of K_s by checking $H(K_s)$ – Hashing via **SHA-1 (Client side)**

Inside int main()

```
cout << "Receiving Encrypted Session Key and Its Hashing Value From Server..." << endl << endl;

string encryptedSessionKey = send_recv(sock, receive, "Received Encrypted Session Key from Server");

send_recv(sock, receive, "Received SHA-1 Hash of Session Key from Server ");

DecryptSession(encryptedSessionKey, privatekey);

verify(SHA1string(encodedSessionKey), hashEncodedSession);

cout << "IDEA KEY : ";
cout << encodedSessionKey << endl << endl;
```

Decrypt the session key with 'RSAES_OAEP_SHA_Decryptor' using the decoded private key and decode the hex and convert it into a string variable.

```
void DecryptSession(string session, string privKey)
{
    string decodedEncHexEnSeshKey;
    StringSource ss(session, true, new HexDecoder(new StringSink(decodedEncHexEnSeshKey)));

    AutoSeededRandomPool rng;
    InvertibleRSAFunction parameters;
    parameters.GenerateRandomWithKeySize(rng, 1024);

    RSA::PrivateKey privateKey(parameters);
    string decodedPrivKey;

    StringSource ss2(privKey, true, (new HexDecoder(new StringSink(decodedPrivKey))));
    StringSource PrivKeySS(decodedPrivKey, true);       //load it into bytes
    privateKey.Load(PrivKeySS);     //load the private key

    RSAES_OAEP_SHA_Decryptor d(privateKey);
    string hexEnSeshkey;
    StringSource ss3(session, true, (new HexDecoder(new PK_DecryptorFilter(rng, d, (new StringSink(hexEnSeshkey))))));
    cout << "Decrypting the Session Key..." << endl;
    cout << "Session Key : " << hexEnSeshkey << endl;
    cout << "SHA-1 Hash Value : " << SHA1string(hexEnSeshkey) << endl;
    cout << "Decryption is complete..." << endl << endl;

    encodedSessionKey = hexEnSeshkey;
}
```

10. Upon completion of Handshake process above, Client is now ready to use K_s to encrypt its data communication **(Client side)**

    The data communication will be done through a do while loop. Firstly the client will enter and encrypt the message.

```
    cout << "IDEA KEY : ";
    cout << encodedSessionKey << endl << endl;

    bool loop = true;
    cin.ignore();
    do
    {
        CFB_IDEA_Encryption(encodedSessionKey, sock);
        CFB_IDEA_Decryption(sock, encodedSessionKey, encryptedmsg);
    } while (loop == true);
}
```

11. Client program: Encrypt the message **Mc** using **IDEA** encryption with the agreed session **Ks** (**IDEA Key size**) **(Client side)**

```
void CFB_IDEA_Encryption(string keys, int sock)
{
    AutoSeededRandomPool prng;
    string decodedkey;
    StringSource s(keys, true, (new HexDecoder(
        new StringSink(decodedkey))
        ) // StreamTransformationFilter
    ); // StringSource

    SecByteBlock key((const byte*)decodedkey.data(), decodedkey.size());

    const byte iv[] = { 0x12,0x34,0x56,0x78,0x90,0xab,0xcd,0xef };
```

12. Message **Mc** enters **IDEA** encryption using **CFB Mode (Client side)**

```
void CFB_IDEA_Encryption(string keys, int sock)
{
    AutoSeededRandomPool prng;
    string decodedkey;
    StringSource s(keys, true, (new HexDecoder(
        new StringSink(decodedkey))
        ) // StreamTransformationFilter
    ); // StringSource

    SecByteBlock key((const byte*)decodedkey.data(), decodedkey.size());

    const byte iv[] = { 0x12,0x34,0x56,0x78,0x90,0xab,0xcd,0xef };

    string plain;
    string cipher, encoded, recovered;

    /*********************************\
    \*********************************/
    do
    {
        cout << "Enter Message to Server : ";
        getline(cin, plain);
        if (plain.size() > 1024)
        {
            cout << "[Client] Message Length is Exceed" << endl;
        }
    } while (plain.size() > 1024);
    try
    {
        CFB_Mode< IDEA >::Encryption e;
        e.SetKeyWithIV(key, key.size(), iv);

        StringSource ss1(plain, true,
            new StreamTransformationFilter(e,
                new StringSink(cipher)
            ) // StreamTransformationFilter
        ); // StringSource
    }
    catch (const CryptoPP::Exception& e)
    {
        cerr << e.what() << endl;
        exit(1);
    }

    StringSource ss2(cipher, true,
        new HexEncoder(
            new StringSink(encoded)
        ) // HexEncoder
    ); // StringSource
    cout << "Cipher Text Entered [HEX Encoded] : " << encoded << endl;
    cout << "[Details] Message is Sent to Server" << endl << endl;
    encryptedmsg = send_recv(sock, encoded, "Received Hex Encoded Message from Server");
}
```

13.Server then Decrypt the message **Mc** from client using **Ks** (**Server side**)

```
    cout << "IDEA Key : " << IDEAkey << endl << endl;
    bool loop = true;
    cin.ignore();
    do {
        cout << "[Server] Receiving/Waiting Message from Client..." << endl;
        CFB_IDEA_msgDecrpytionEncryption(new_socket);
    } while (loop == true);
    return 0;
```

```
void CFB_IDEA_msgDecrpytionEncryption(int new_socket)
{
    CFB_IDEA_Decryption(new_socket, IDEAkey);
    CFB_IDEA_Encryption(IDEAkey, new_socket);
}
```

```
void CFB_IDEA_Decryption(int socket, string keys)
{
    int valread;
    char buffer[1024] = { 0 };
    valread = read(socket, buffer, 1024);
    cout << "Cipher Text from Client [HEX Encoded] : " << buffer << en
    AutoSeededRandomPool prng;
    string rawcipher, decodedkey;
    StringSource ss2(buffer, true,
        new HexDecoder(
            new StringSink(rawcipher)
        )  //HexEncoder
    ); // StringSource

    StringSource s(keys, true, (new HexDecoder(
        new StringSink(decodedkey))));
    SecByteBlock key((const byte*)decodedkey.data(), decodedkey.size()
    const byte iv[] = { 0x12,0x34,0x56,0x78,0x90,0xab,0xcd,0xef };

    try
    {
        CFB_Mode< IDEA >::Decryption d;
        d.SetKeyWithIV(key, key.size(), iv);
        string recovered;

        StringSource ss3(rawcipher, true,
            new StreamTransformationFilter(d,
                new StringSink(recovered)
            ) // StreamTransformationFilter
        ); // StringSource
        if (recovered == "quit")
        {
            cout << "Program Quit..." << endl << endl;
            sendpacket(socket, buffer);
            exit(1);
        }
        cout << "Decrypted Text : " << recovered << endl << endl;
    }
    catch (const Exception& e)
    {
        cerr << e.what() << endl;
        exit(1);
    }
}
```

14. Server program: Encrypt the message **Ms** using **IDEA** encryption with the agreed session **Ks** (**IDEA Key size**) (**Server side**)

```
string CFB_IDEA_Encryption(string IDEAkey, int sock)
{
    AutoSeededRandomPool prng;
    string decodedkey;
    StringSource s(IDEAkey, true, (new HexDecoder(
        new StringSink(decodedkey))
        ) // StreamTransformationFilter
    ); // StringSource

    SecByteBlock key((const byte*)decodedkey.data(), decodedkey.size());
    const byte iv[] = { 0x12,0x34,0x56,0x78,0x90,0xab,0xcd,0xef };
```

15. Message **Ms** enters **IDEA** encryption using **CFB Mode** (**Server side**)

```
    SecByteBlock key((const byte*)decodedkey.data(), decodedkey.size());
    const byte iv[] = { 0x12,0x34,0x56,0x78,0x90,0xab,0xcd,0xef };

    string plain;
    string cipher, encoded, recovered;

    /*********************************\
    \*********************************/

    do {
        cout << "Enter Message to Client : ";
        getline(cin, plain);
        if (plain.size() > 1024)
        {
            cout << "[Server] Message Length is Exceed" << endl << endl;
        }
    } while (plain.size() > 1024);
    try
    {
        CFB_Mode< IDEA >::Encryption e;
        e.SetKeyWithIV(key, key.size(), iv);

        StringSource ss1(plain, true,
            new StreamTransformationFilter(e,
                new StringSink(cipher)
            ) // StreamTransformationFilter
        ); // StringSource
    }
    catch (const Exception& e)
    {
        cerr << e.what() << endl;
        exit(1);
    }

    StringSource ss2(cipher, true,
        new HexEncoder(
            new StringSink(encoded)
        ) // HexEncoder
    ); // StringSource
    cout << "Cipher Text Entered [HEX Encoded] : " << encoded << endl;
    sendpacket(sock, encoded);
    cout << "[Details] Message is Sent to Client " << endl << endl;
    return plain;
}
```

16. Communication continues until '**Quit'** signal initiated (**Server side**)

```
CFB_Mode< IDEA >::Decryption d;
d.SetKeyWithIV(key, key.size(), iv);
string recovered;

StringSource ss3(rawcipher, true,
    new StreamTransformationFilter(d,
        new StringSink(recovered)
    ) // StreamTransformationFilter
); // StringSource
if (recovered == "quit")
{
    cout << "Program Quit..." << endl << endl;
    sendpacket(socket, buffer);
    exit(1);
}
```

17. Upon receiving message **Ms** from server, client can decrypt it using the session **Ks**.**(Client side)**

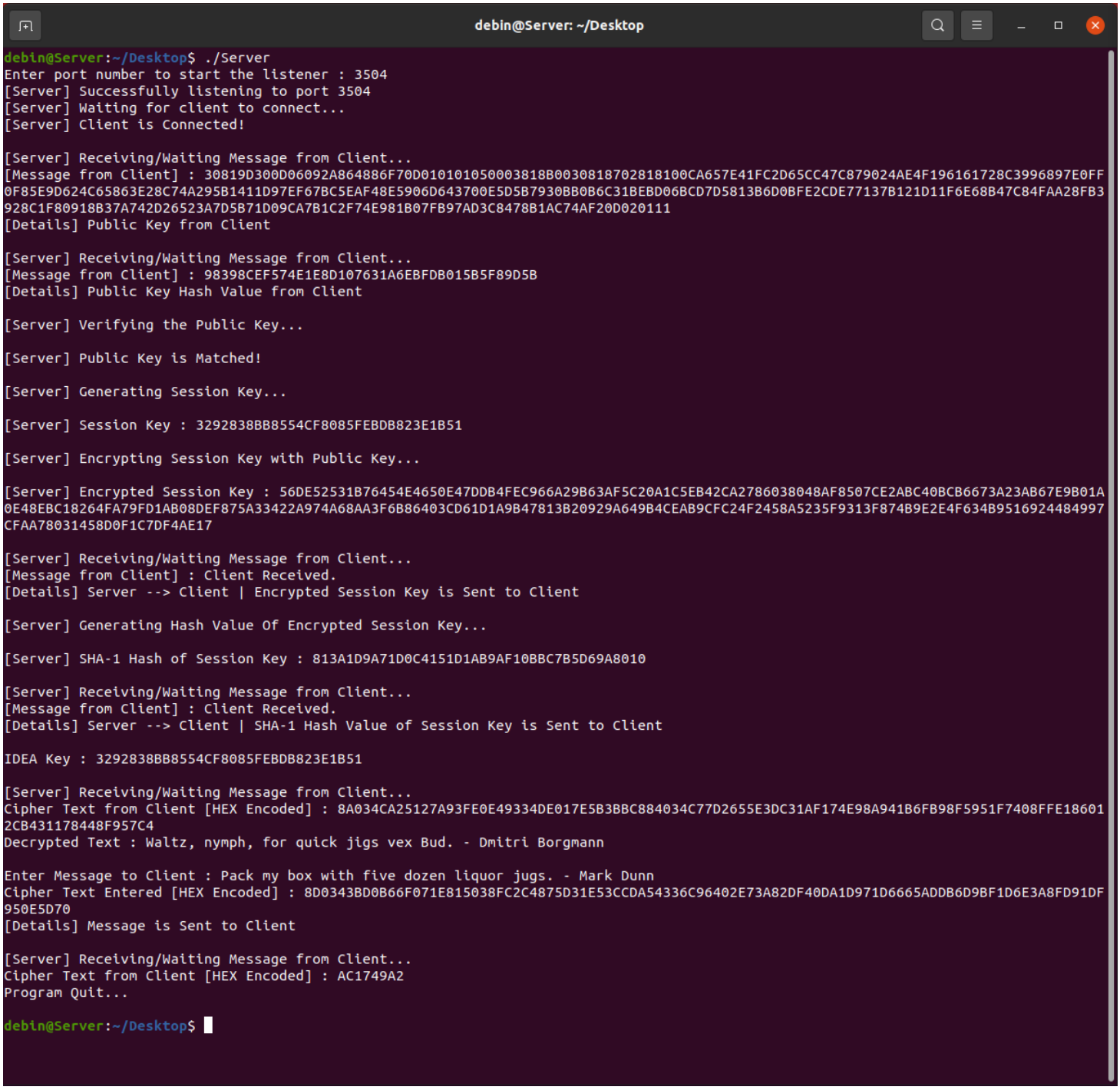
```
void CFB_IDEA_Decryption(int socket, string keys, string encryptedmessage)
{
    AutoSeededRandomPool prng;
    string rawcipher, decodedkey;
    StringSource ss2(encryptedmessage, true,
        new HexDecoder(
            new StringSink(rawcipher)
        ) // HexEncoder
    ); // StringSource

    StringSource s(keys, true, (new HexDecoder(
        new StringSink(decodedkey))));

    SecByteBlock key((const byte*)decodedkey.data(), decodedkey.size());
    const byte iv[] = { 0x12,0x34,0x56,0x78,0x90,0xab,0xcd,0xef };

    try
    {
        CFB_Mode< IDEA >::Decryption d;
        d.SetKeyWithIV(key, key.size(), iv);
        string recovered;

        StringSource ss3(rawcipher, true,
            new StreamTransformationFilter(d,
                new StringSink(recovered)
            ) // StreamTransformationFilter
        ); // StringSource
        if (recovered == "quit")
        {
            cout << "Program Quit..." << endl << endl;
            sendpacket(socket, encryptedmessage);
            exit(1);
        }

        cout << "\x1b[A" << "Decrypted Text : " << recovered << endl << endl;
    }
    catch (const CryptoPP::Exception& e)
    {
        cerr << e.what() << endl;
        exit(1);
    }
}
```

18.Communication continues until ‘**Quit’** signal initiated **(Client side)**

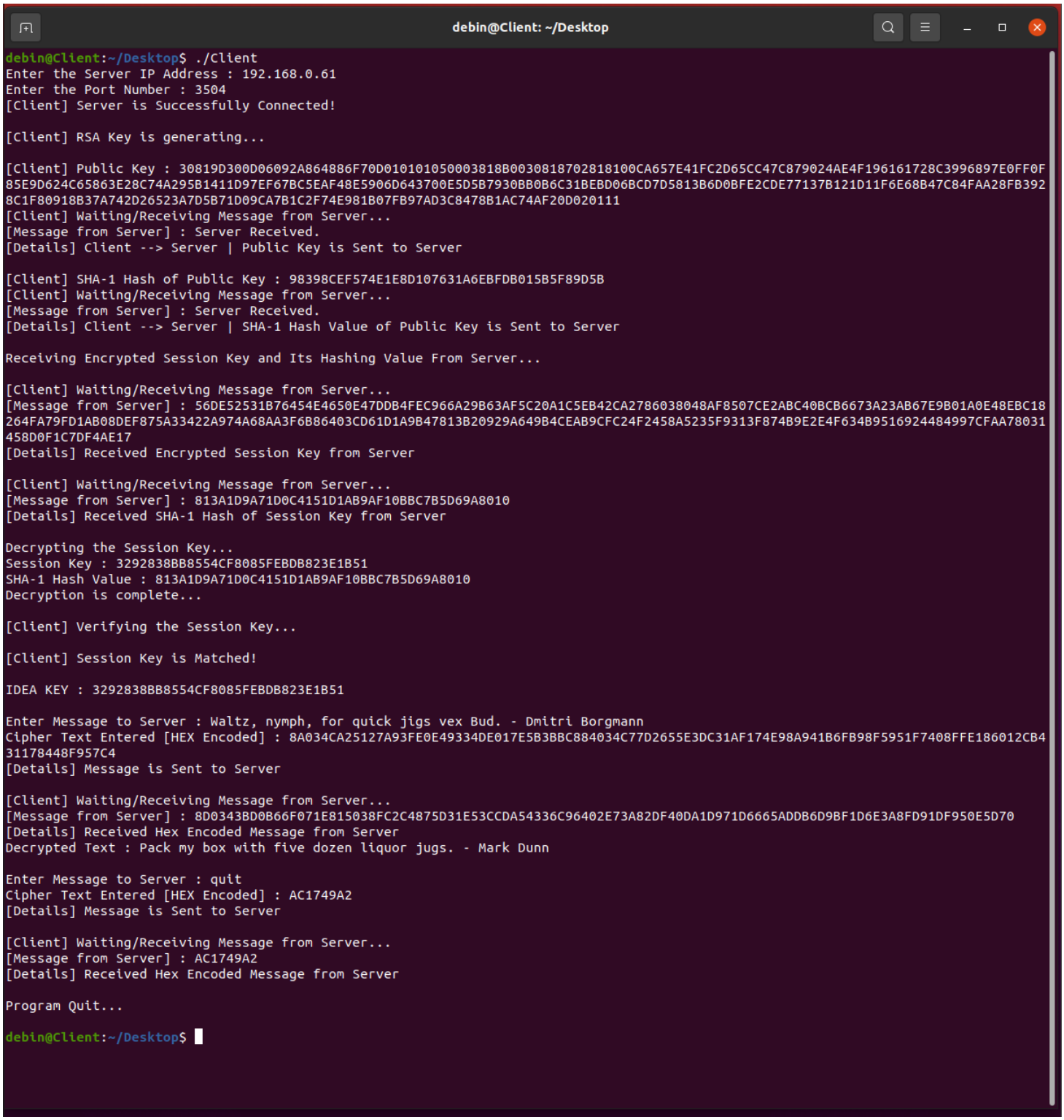
```
CFB_Mode< IDEA >::Decryption d;
d.SetKeyWithIV(key, key.size(), iv);
string recovered;

StringSource ss3(rawcipher, true,
    new StreamTransformationFilter(d,
        new StringSink(recovered)
    ) // StreamTransformationFilter
); // StringSource
if (recovered == "quit")
{
    cout << "Program Quit..." << endl << endl;
    sendpacket(socket, encryptedmessage);
    exit(1);
}
```

Program Screenshots

(Server Machine)



(Client Machine)



Compilation

Linux OS (Ubuntu)

(Server)

1. Open terminal
2. Run the following commands:
3. sudo apt-get update
4. sudo apt-get install libcrypto++-dev libcrypto++-doc libcrypto++-util (Install Crypto++® Library 8.5)
5. cd Desktop (cpp file location in my case is Desktop)
6. g++ -g3 -ggdb -O0 -Wall -Wextra -Wno-unused -o Server server.cpp -lcryptopp
7. ./Server

(Client)

1. Open terminal
2. Run the following commands:
3. sudo apt-get update
4. sudo apt-get install libcrypto++-dev libcrypto++-doc libcrypto++-util (Install Crypto++® Library 8.5)
5. cd Desktop (cpp file location in my case is Desktop)
6. g++ -g3 -ggdb -O0 -Wall -Wextra -Wno-unused -o Client client.cpp -lcryptopp
7. ./Client